ASC-TR-93-5008

AN INTRODUCTION TO STRUCTURAL MODELS

JOE BATMAN SOFTWARE ENGINEERING INSTITUTE
LARRY HOWARD SOFTWARE ENGINEERING INSTITUTE
BILL SCHELKER AERONAUTICAL SYSTEMS CTR

SOFTWARE ENGINEERING INSTITUTE
CARNEGIE-MELLON UNIVERSITY
PITTSBURGH, PA 15213

AUGUST 1992

FINAL REPORT FOR JUNE 1986 TO AUGUST 1992

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION IS UNLIMITED.

SYSTEMS ENGINEERING PROCESS DIVISION
AERONAUTICAL SYSTEMS CENTER
AIR FORCE MATERIEL COMMAND
WRIGHT PATTERSON AFB OH 45433-7126

NOTICE

When Government drawings, specifications, or other data are used for any purpose other than in connection with a definitely Government-related procurement, the United States Government incurs no responsibility or any obligation whatsoever. The fact that the government may have formulated or in any way supplied the said drawings, specifications, or other data, is not to be regarded by implication, or otherwise in any manner construed, as licensing the holder, or any other person or corporation; or as conveying any rights or permission to manufacture, use, or sell any patented invention that may in any way be related thereto.

This report is releasable to the National Technical Information Service (NTIS). At NTIS, it will be available to the general public, including foreign nations.

This technical report has been reviewed and is approved for publication.

JEFFREY G. VALITON, Maj, USAF
Program Manager
Systems Engineering Process Division

James D Basinger
JAMES D. BASINGER
Chief, Tech Development and Insertion
Systems Engineering Process Division

James J O'Connell
JAMES J. O'CONNELL
Chief, Systems Engineering Process Division
Training Systems Program Office

If your address has changed, if you wish to be removed from our mailing list, or if the addressee is no longer employed by your organization please notify ASC/YTED , WPAFB, OH 45433-7111 to help us maintain a current mailing list.

Copies of this report should not be returned unless return is required by security considerations, contractual obligations, or notice on a specific document.

REPORT DOCUMENTATION PAGE

AUG 1992

3. REPORT TYPE AND DATES COVERED
FINAL 06/86 to 08/92

AN INTRODUCTION TO STRUCTURAL MODELS

C F19628-90-C-0003
PE 64227
PR 2325
TA
WU

JOE BATMAN SOFTWARE ENGINEERING INSTITUTE
LARRY HOWARD SOFTWARE ENGINEERING INSTITUTE
BILL SCHELKER AERONAUTICAL SYSTEMS CTR

SOFTWARE ENGINEERING INSTITUTE
CARNEGIE-MELLON UNIVERSITY
PITTSBURGH, PA 15213

8. PERFORMING ORGANIZATION REPORT NUMBER

SYSTEMS ENGINEERING PROCESS DIVISION
AERONAUTICAL SYSTEMS CENTER
AIR FORCE MATERIEL COMMAND
WRIGHT PATTERSON AFB OH 45433-7126

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
ASC-TR-93-5008

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION IS UNLIMITED.

THIS PAPER INTRODUCES STRUCTURAL MODELS FOR FLIGHT SIMULATORS TO TECHNICAL MANAGERS. IT GIVES THE RATIONALE FOR USING STRUCTURAL MODELS, DEFINES STRUCTURAL MODELS, AND DISCUSSES PRACTICES ASSOCIATED WITH THEIR USE. THE PAPER PROVIDES AN EXAMPLE OF A STRUCTURAL MODEL BASED ON THE MODEL USED ON A NUMBER OF RECENT SIMULATOR ACQUISITIONS. THE PAPER ALSO DISCUSSES WHAT ASC/YT EXPECTS IN PROPOSALS FOR THE USE OF STRUCTURAL MODELS IN THE DEVELOPMENT OF TRAINING SIMULATORS.

PARTITIONING
INTENDED QUALITIES

NUMBER OF PAGES 30

UNCLASSIFIED UNCLASSIFIED UNCLASSIFIED UL

Table of Contents

List of Figures

FOREWORD

This paper is a technical deliverable by the Software Engineering Institute (SEI), Carnegie Mellon University, to the Training System Program Office, ASC/YT, Wright Patterson Air Force Base, in support of an ASC/YT program to transition technology and use of structural models in training simulators. The paper describes key concepts of structural models and contains general guidance for technical proposals. Program specific details of proposal requirements are in the Instructions to Offerors.

An Introduction to Structural Models

Abstract: This paper introduces structural models for flight simulators to technical managers. It gives the rationale for using structural models, defines structural models, and discusses practices associated with their use. The paper provides an example of a structural model based on the model used on a number of recent simulator acquisitions. The paper also discusses what ASC/YT expects in proposals for the use of structural models in the development of training simulators.

1 Introduction

This paper has been placed in the bidder's library to introduce structural models to technical managers. It introduces key concepts and describes what ASC/YT expects to see in proposals for the use of structural models in the development of **flight simulators**. ASC/YT intends to follow this with two other activities:

1. Acquisition practices will be modified to accomodate the design issues addressed by structural models, both in the evaluation of proposals and the management of acquisitions.
2. A guidebook for engineers will be written that will provide details on the most successful structural models and the development practices associated with them.

Structural models are a software engineering tool for the system engineering team. ASC/YT began work on structural models in 1986, in response to a number of influences. The first was a realization that as the scale and complexity of simulators increased, the traditional architectures for flight simulators were reaching their limits. In particular, modifiability was becoming increasingly difficult. At the same time, concurrency between the simulator and the aircraft being simulated was becoming increasingly important to ASC/YT. Also at the same time, Ada and **object-oriented** notions were being investigated for their applicability to real-time flight simulators. All of these factors contributed to the development and ongoing evolution of designs that were responsive to the issues of increasing scale, concurrency, and an appropriate use of Ada for real-time simulation.

Structural models have evolved through use on a number of recent simulator acquisitions including the B-2 Weapon Systems Trainer, the C-17 Aircrew Training System, and the Special Operations Forces Aircrew Training System programs. These programs have demonstrated to ASC/YT that structural models provide a viable basis for describing, evaluating, and developing software for large, complicated flight simulators. Benefits that have resulted from the use of structural models include:

- The ability to make system engineering decisions at an early stage in the system's evolution.
- The ability to evaluate the effects of these decisions early in the development of a system.
- The ability to consistently enforce these decisions across large simulation projects.

This paper introduces contractors to the key concepts of structural models. It does not advocate the use of a particular structural model. There are two reasons why ASC/YT will not dictate a particular structural model:

1. A single structural model is not sufficient for all aspects of a flight simulator. Air vehicle systems, the instructor/operator station (IOS), the tactical environment, and mission planning all represent different problems that require different structural models.
2. Structural models for air vehicle systems, which represent the most advanced use of structural models, are still evolving. Each program has improved what has been accomplished by previous programs, and ASC/YT expects improvements to continue.

This paper does not contain enough detail to train a contractor to use structural models. To that end, ASC/YT is commissioning an engineering guidebook on structural models and development practices. This paper also does not fully explore the relationship between structural models and other software engineering technologies, such as tools and design methodologies. Please note, however, that structural models are fully consistent with the principles of software engineering and do not represent an attempt to introduce a radical new technology to the flight simulator community.

The body of this paper consists of two chapters on structural models and a chapter on proposals for their use.

Chapter 2 defines the term *structural model* and elaborates on the definition. It talks in general terms about the relationship between structural models and software quality, and the application of a structural model to a project.

Chapter 3 describes the structural model used in a number of recent simulator acquisitions, a structural model that is responsive to certain quality goals and is characterized by certain development practices.

Chapter 4 discusses what ASC/YT expects of proposals for the use of structural models in the development of flight simulators.

In addition, we provide a glossary, **Appendix A,** which defines terms used in this paper. Terms that appear in the glossary are printed in boldface when we first introduce them in the text.

2 Concepts of Structural Models

A **structural model** is a pattern for specifying and implementing software system functionality. It reflects system engineering decisions about partitioning and coordination. A structural model allows critical decisions about system design to be evaluated early in development and allows the decisions to be applied consistently across the entire software system.

In the remainder of this chapter, we elaborate on this definition. Sections 2.1 and 2.2 explain what we mean by partitioning and coordination and discuss the role of system engineering in making decisions about these issues. Section 2.3 discusses evaluation of a structural model, and Section 2.4 discusses how to apply a structural model consistently on a project. Because structural models have thus far been primarily used on simulation of the air vehicle portion of flight simulators, we draw the discussions and examples from that domain.

2.1 Partitioning Strategies and Software Structure

Problems such as simulating the air vehicle of a modern aircraft are often too large and complex to be understood and solved as a whole. For this reason, engineers partition, or decompose, such problems into smaller subproblems that are more easily understood and solved. The solutions to these smaller subproblems together constitute a solution to the overall problem. This process is depicted in Figure 2-1.

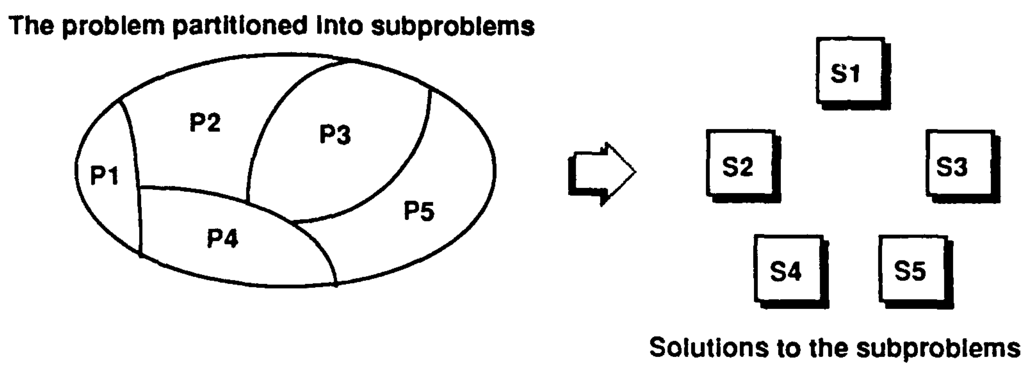

Figure 2-1: Dividing and Conquering a Problem Through Partitioning

Partitioning is the decomposition of large software problems into smaller subproblems and the allocation of these subproblems to software components that will solve them. This decomposition establishes interfaces between the software components and creates the need to coordinate the **computations** of the software components in order to form the overall solution.

Partitioning involves making decisions about how the problem will be decomposed and allocated to software components. Partitioning strategies guide this decision making. In any system, several partitioning strategies are possible. For simulations of air vehicles, partitioning could be based on:

- Training tasks
- Pilot tasks
- Air vehicle components
- The closing of feedback loops.

In the next chapter, we discuss a structural model based on a partitioning strategy of capturing individual air vehicle components in individual software components.

Different partitioning strategies lead to different software structures. This is depicted in Figure 2-2.

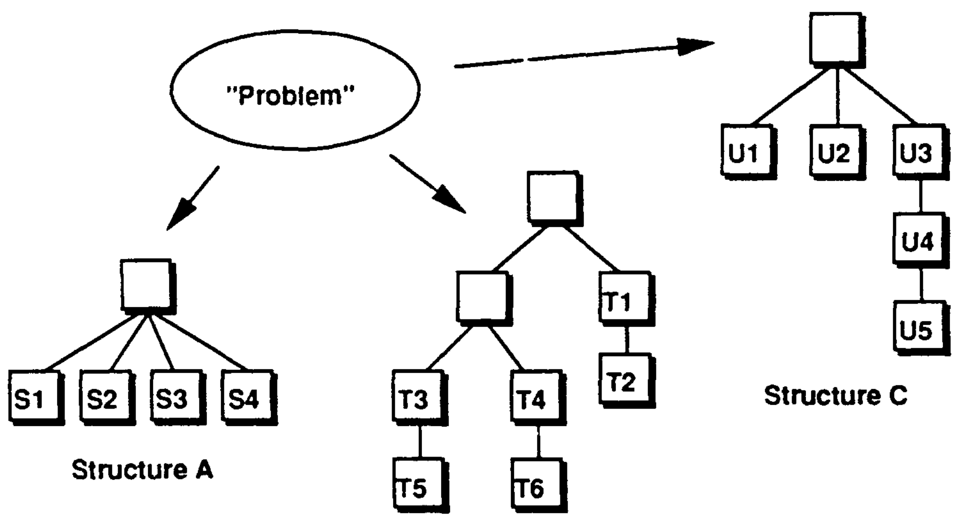

Figure 2-2: Different Structures for Solving the Same Problem

The alternative software structures in Figure 2-2 all have the same functionality. How, then, do we determine which software structure is better? The answer depends in large part on the non-functional qualities or attributes that the system is intended to exhibit. Qualities such as modifiability, efficiency, and reusability are non-functional in the sense that they are not part of a system's functionality; nevertheless, they are capabilities that are important to the system. In this paper, we will refer to these qualities as **intended qualities** of the system. A few dominant qualities tend to guide decision making in the design of large software systems. These qualities may be realized more easily through one software structure (and, therefore, as a result of one partitioning strategy) than another.

System engineers partition system requirements into software components. Decisions about which functions are partitioned into which elements are guided by a partitioning strategy, experience, and consideration of the intended qualities that the system is to exhibit. The process of partitioning is iterative. Each attempt at partitioning produces a hypothesis that is evaluated. An initial partitioning is often unacceptable, leading to an alternative partitioning that also is evaluated. Eventually, the process leads to an acceptable partitioning.

The result of this partitioning is a software structure that specifies the functionality of each software component. But system engineering considerations remain. We have discussed decomposition (partitioning). We now turn our attention to recomposition, or how the individual software components will be integrated to form the complete solution. We will look at how the software components work together to satisfy the overall functional requirements.

2.2 Coordination

In the previous section, we described how total system functionality is partitioned into software components according to some strategy. Once functionality has been partitioned, issues of coordination must be considered. The result of this consideration is a collection of **structural elements** that embody decisions about how to coordinate the behavior of the individual components to satisfy the overall system functional requirements. We will now elaborate on the main issues of coordination—communication and activation.

Communication refers to the paths and mechanisms by which structural elements share the results of their computations. For example, elements may communicate by way of parameter passing or shared memory. The communication mechanisms that affect each structural element must be explicitly stated. Also, the other structural elements with which each structural element directly communicates and the direction of communication must be made explicit. Given two structural elements, A and B, it must be possible to determine that A furnishes data that B uses, B furnishes data that A uses, or neither.

Activation refers to the mechanisms by which the structural elements are made to cooperate to realize overall system functionality. This includes consideration of:

- How the various elements are scheduled to perform their computations.
- How they are notified of events (for example, malfunction setting/clearing, **simulator mode** changes, etc.).
- How the computations on one processor are coordinated with computations on another processor when more than one processor is used.

Decisions about coordination mechanisms affect both the specification and the implementation of software developed using a structural model. The coordination mechanisms dictate what aspects of system behavior must be specified for each partitioned component. For example, if there is a mechanism for setting malfunctions, then the specification for each subsystem and component must describe malfunction behavior. If the overall system is to function in freeze mode, then the freeze mode behavior of each subsystem must be specified.

The coordination mechanisms provide a means of generalizing about the use of programming language features across partitions. For example, system engineers must decide how to implement the structural element that allows activation of the update mode functionality of each subsystem. The system engineering decisions about the form the structural elements take in the implementation resolves the tradeoff between conflicting goals such as the benefits of encapsulation and the overhead of communicating hidden information. For example, a decision

to package components as private Ada types requires mechanisms to access component state. The system engineer's goal, therefore, is to arrive at mutually consistent coordination decisions that are also consistent with the goals of the partitioning strategy.

To achieve this goal, the engineer must generalize. The air vehicle simulation for a modern aircraft can be partitioned into 200 or more subsystems, and each subsystem can be further decomposed into one or more components, which can themselves be further decomposed. But the system engineer need not independently hypothesize and evaluate coordination decisions for each of the many subsystems. Instead, the system engineer can hypothesize a set of decisions for one subsystem, regardless of the specific functionality performed by the subsystem. If the decisions appear plausible, the system engineer can imagine a "complete system" in which all the subsystems are packaged identically.The same coordination mechanisms can then be used, regardless of the specific functionality performed by each subsystem. This complete system is then the object of evaluation.

The system engineer repeats this process of generating and evaluating hypotheses until an acceptable set of decisions about coordination is reached. *Structural model* is the name given to this acceptable set of decisions. *Structural* refers to the fact that the system engineer is concerned with a software structure that describes how components are activated and how they communicate. *Model*, in this context, refers to two facts:

1. To make decisions about partitioning and coordination for one subsystem is to make those decisions for all subsystems.
2. The structural model is a *model* of the system that is to be built. Decisions about structural elements in particular allow one to reason about the structural model as though it were the eventual software system itself.

Thus, a structural model reflects system engineering decisions about partitioning and coordination. A structural model allows early evaluation of the system that is to be built and provides for the consistent specification and implementation of air vehicle functionality. These are the subjects of the next two sections.

2.3 Evaluation

A structural model is a model of the system that is to be built. As such, it can be evaluated in two ways:

1. How well it represents the system to be built.
2. How well the system to be built will satisfy the required functionality and the intended qualities.

In this section, we discuss both of these types of evaluation.

2.3.1 Evaluation of the Structural Model

The purpose of a structural model is to convey information about the system to be built. The following are criteria for how well it achieves this purpose:

- *Does it include an explicit statement of the partitioning strategy?* The partitioning strategy used at the various levels of the structural model must be made explicit.
- *Is the software structure suitable for evaluation?* The ability to evaluate a proposed software structure depends on the number of different structural elements whose interactions must be analyzed. This number should be small enough that the interactions of the structural elements can be evaluated.
- *Is the software structure a suitable guide for system development?* The structure must be suitable for elaboration into a system that computes the required functionalities and has the intended qualities.
- *Are the coordination mechanisms clearly delineated and non-redundant?*
- *Can the final system be consistently rendered from the structural model?*

2.3.2 Evaluation of the System that Will Be Built

One of the key benefits of using a structural model is that it enables the system that will be built from it to be evaluated at an early stage in the system's development. A system can be evaluated with respect to:

- Functionality
- Intended qualities

When examining system functionality, it is useful to distinguish between two broad categories of simulation requirements:

1. Requirements to simulate aircraft systems and effects, such as radar and ground effects.
2. Requirements that span the simulation of individual aircraft systems, such as the ability to insert malfunctions, to record and playback segments of a training mission, to reconfigure the training mission, and to initialize to a particular training mission.

A structural model must permit all such requirements to be satisfied and must allow the simulator to operate in all required modes. Further, by examining the structural elements (the results of the partitioning) and the coordination mechanisms that have been established, it should be evident from the structural model exactly how these requirements will be satisfied in the final system.

Because a few dominant intended qualities often guide decision making in the design of large software systems, it is important for these qualities to be made explicit from the outset of design. Evaluation of a structural model with respect to the system's intended qualities can be difficult because direct measurement is seldom possible, particularly during early

development. Nevertheless, a structural model should make it possible early in development to envision a complete software system, built from the structural elements, that employs the specified coordination mechanisms. It should then be possible to reflect on specific intended qualities and judge the responsiveness of the structural model. For example, if concurrency between simulator and aircraft is a goal, one can test the responsiveness of a structural model to some anticipated changes in the aircraft being simulated. Such tests can reveal weaknesses in the partitioning strategy or coordination mechanisms that limit flexibility.

2.4 Application of a Structural Model

A structural model reflects decisions about partitioning and coordination mechanisms. After the system engineer has made these decisions and analyzed their appropriateness, the goal in applying a structural model on a project is to see that those decisions are adhered to consistently. Consistent use of the structural model must be explicitly and rigorously enforced, particularly in large simulation projects whose project teams are geographically and organizationally distributed.

There are two aspects of consistency in applying a structural model:

1. Consistent use of the structural elements by the analysts.
2. Consistent transformations between the specification and the implementation.

To achieve consistent use of the structural elements, system engineers must be sure that the partitioning strategy is clearly and explicitly stated. The analysts responsible for the specific subsystems must understand the partitioning strategy and must understand how the elements are expected to coordinate with other elements.

Consistent transformations can be achieved through the use of **specification forms** and **code templates**. Specification forms can be used by simulation analysts to specify how individual subsystems will be simulated in terms of the structural model. That is, a specification form can constrain the simulation analyst to express solutions to simulation problems in terms of the coordination mechanisms contained in the structural model.

Code templates can be used to ensure the use of consistent coordination mechanisms. The relationship between a specification form and a set of code templates is a straightforward mapping. The specification form and the code templates refer to the same structural elements and coordination mechanisms. Therefore, the implementation of a specification that calls for the use of structural element X would include instantiation of the code template for structural element X.

Consistent application of a structural model can lead to certain development efficiencies and can contribute to predictability with respect to computer and human resources. The efficiencies result in large part from reuse of the design decisions reflected in the structural model. Important decisions about partitioning and coordination mechanisms are made once

and are reflected in the specification forms that simulation analysts use. Thus, rather than deferring coordination decisions and making them repetitively for different subsystems, simulation analysts make the decisions once and use these decisions consistently throughout the system. A structural model also offers the potential for reuse of software structures from project to project.

Predictability results from the consistent application of the structural model. In the application of a structural model, many simulation analysts repeatedly apply the same structural elements and coordination mechanisms to the simulation of many different subsystems. Therefore, the lessons learned about expenditure of effort and use of computer resources while developing one subsystem can be applied to the development of subsequent subsystems.

2.5 Summary

A structural model is a high-level software design that explicitly reflects decisions about partitioning and coordination. When these decisions are made early in the development process, they enable solutions to the problems of partitioning and coordination to be made once and applied uniformly throughout the entire system.

Since many possible decisions about partitioning and coordination are possible, many different structural models for flight simulators are possible. The most appropriate structural model in a particular context is the one that is:

- Mapped most easily to required system functionality.
- Most able to provide intended qualities.
- Of acceptable cost.

Thus, because the functionality delivered by an instructor/operator station (IOS) is significantly different from that delivered by an air vehicle simulation, a structural model for IOS software may be different from a structural model for air vehicle simulation software. Further, structural model "goodness" is always relative to functional requirements and intended qualities. When certain qualities are emphasized over others, one structural model is more suitable than others.

The most significant benefits of a structural model come from its consistent application on a project. Consistent use of a structural model becomes a greater challenge as development teams grow larger and more geographically or organizationally distributed. Specification forms and code templates for simulation analysts offer one way of enforcing consistent use of a structural model. Consistency leads to efficiency in the reuse of design decisions and predictability with respect to the use of human and computer resources.

3 The Air Vehicle Structural Model

Traditional flight simulators emphasized computer **efficiency**. The qualities required by the simulators of the past, however, are different from the qualities required by the simulators of today. Increases in available computing power and aircraft complexity have led to a gradual shift in emphasis away from computer efficiency alone toward a more ambitious set of goals that includes modifiability and scalability in addition to efficiency. The actions that ASC/YT is taking to institutionalize structural models (see page 2) result from the conviction that modifiability and scalability are important qualities for modern-day flight simulators.

Modifiability is the ability to change software components in a simulation easily. Modifiability is important because of the need for concurrency between the simulator and the aircraft being simulated. **Scalability** is the ability of a software system to accommodate changes in scale. Scalability is important because, as aircraft become increasingly sophisticated, the size of the simulators being built, in terms of lines of code and number of aircraft subsystems being simulated, is increasing.

In this chapter, we present an air vehicle structural model that is the product of an emphasis on modifiability and scalability as well as efficiency. The essential structural elements of this structural model are depicted in Figure 3-1. In the remainder of this chapter, we will elaborate on the partitioning strategy and coordination and communication mechanisms reflected in the structural elements. We will then evaluate the air vehicle structural model and discuss several aspects of its use.

3.1 Partitioning Strategy

The air vehicle structural model is based on an object-oriented partitioning strategy in which aircraft components form the basis of the software structure. Aircraft systems are decomposed into subsystems, and subsystems are decomposed into a collection of **objects** that correspond to real-world aircraft components like reservoirs, motor pumps, relief valves, etc. The purpose of an object is to simulate the behavior of an individual aircraft component. When decomposing a subsystem into objects, **fidelity** and training requirements play a large role in determining which objects to include in the simulation of each specific subsystem.

Systems that are unique to the simulation, such as aerodynamics, equations of motion, etc., are also decomposed into subsystems and objects in the same manner as aircraft systems. Requirements for operational modes and other training requirements—run, freeze, mission initialization, malfunctions—are allocated to subsystems and objects. This is consistent with the overall object-oriented philosophy in which objects encapsulate everything there is to know about a real-world component and provide operations, or methods, for making the objects behave in a particular way. For example, the requirement to initialize a mission is allocated across all subsystems and components; all objects are responsible for initializing themselves.

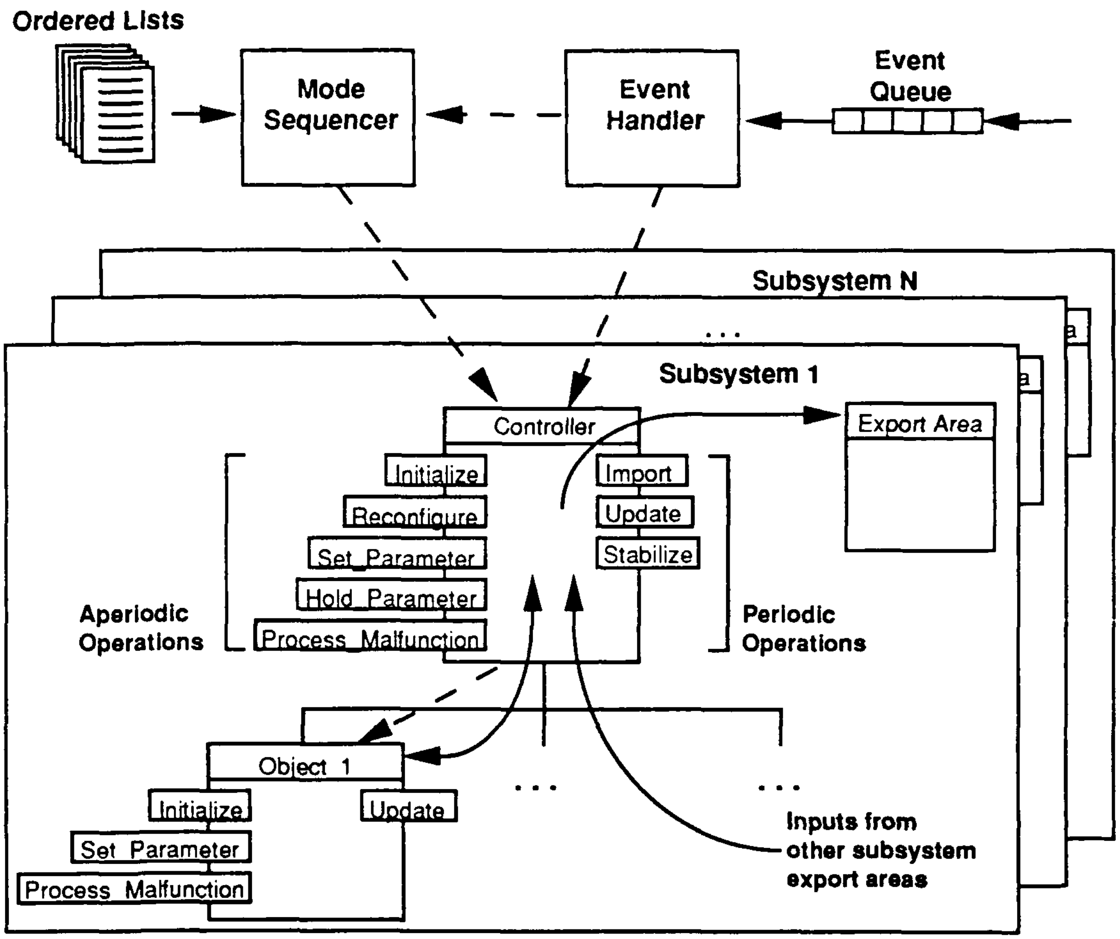

Figure 3-1: Structural Elements of the Air Vehicle Structural Model

3.2 Coordination

All objects, independent of the aircraft component they simulate, are consistently realized in software as Ada packages with private internal states and an identical set of procedures that operate on the objects. Operations on objects fall into two categories: those that are performed **periodic**ally (such as *update*) and those that are performed **aperiodic**ally (such as *process malfunction* or *set parameter*).

Subsystem controllers coordinate the computations performed by individual objects to yield meaningful subsystem behavior. For example, a hydraulics subsystem controller might coordinate the computations of a reservoir object, a motor pump object, and a relief valve object in such a way that, taken together, they yield the behavior of a hydraulics subsystem. Thus, the behavior of an aircraft subsystem is simulated by simulating and coordinating the behaviors of its components. This is different from the way simulators were designed in the past. In the past, a subsystem was just a logical arrangement of modules. In the air vehicle structural model, the subsystem manifests itself as an arrangement of objects under the control of a subsystem controller.

The kind of knowledge that a subsystem controller uses to coordinate the computations of its objects is typically too sophisticated to be represented in a simple table that captures the update order for its objects. Depending on the mode of the simulator or malfunctions that are in effect, a subsystem controller may use its objects in various ways to produce new values. For example, updating a subsystem may involve updating the same object several times during a single frame or not updating an object at all during certain frames. All subsystem controllers, independent of the objects they coordinate, are consistently realized in software as Ada packages with private internal states and an identical set of procedures that operate on the controllers. Like operations on objects, operations on subsystem controllers are either periodic or aperiodic.

Export areas communicate the results of one subsystem's computations with other subsystems. There is exactly one export area associated with each subsystem controller. What is exported by one subsystem is determined by the needs of other subsystems. Since objects are packaged to be independent of specific data sources and sinks, objects do not access export areas directly. Instead, subsystem controllers localize the knowledge of where to find inputs and where to place outputs for the subsystem. Inputs and outputs of objects are passed by parameter; inputs and outputs of subsystems are held in export areas. All export areas are consistently realized in software as simple Ada data packages.

Aperiodic coordination is effected by an **event handler**. The event handler detects and responds to events that originate outside the air vehicle, such as from the instructor/operator station (IOS). Typical events include simulator mode changes such as freeze, run, re-initialize, etc.; malfunction setting and clearing; and parameter setting or holding. An event queue is used to notify the air vehicle that an event has occurred. The event handler does not respond to events directly; instead, it simply dispatches the events to the subsystem controller(s) that require notification. From there, the subsystem controller(s) forward the event to affected objects. Thus, an event handler's knowledge of events is limited to a mapping from a set of events to a set of subsystem controllers to notify. This mapping is realized as a data structure that may be read from disk during simulator initialization. The event handler is realized in software as an Ada package.

Periodic activation is effected by a **mode sequencer**. Just as subsystem controllers coordinate the behavior of objects to yield subsystem behavior, the mode sequencer activates subsystem controllers to yield overall simulator behavior in all required modes. The mode sequencer's visibility is limited to subsystem controllers; it is not aware of the specific objects that comprise a subsystem. Each operational mode of the simulator—initialization, run, freeze, reconfigure, etc.—can be described in terms of an ordered list of subsystem controllers and operations (*initialize, update, stabilize,* etc.) to be performed on each. These lists are realized as data structures that may be read from disk during simulator initialization. The mode sequencer can also provide time-consistent views of data to subsystems that require them by calling operations that import subsystems while the simulation is quiescent. The mode sequencer is realized in software as an Ada package.

In a multiprocessor hardware configuration, each processor would contain the structural elements arranged as described above. That is, each processor would contain a number of subsystems (subsystem controllers, objects, and export areas), an event handler, and a mode sequencer. Depending on the hardware configuration, communication between subsystems on different processors can be accomplished in a variety of ways. For example, if processors do not share memory but are connected by a network, data gathering and distribution subsystems can be implemented. If processors do share memory, it may be sufficient to locate all or some export areas in memory shared between processors.

Although this description of the air vehicle structural model is necessarily incomplete, it should be sufficient for the purposes of this paper. A thorough description will be provided in the forthcoming guidebook on structural models.

3.3 Evaluation of the Air Vehicle Structural Model

We stated earlier that a system that will be built from a structural model can be evaluated with respect to its ability to deliver functional requirements and intended qualities. Although a thorough evaluation of a system built from the structural model described above is beyond the scope of this paper, we will do two things in this section:

1. We will argue that the air vehicle structural model can be used to partition all of the functionality within its scope (air vehicle simulation) and that the coordination mechanisms are suitable for all modes of simulator operation.
2. We will illustrate the effects of the partitioning strategy and coordination mechanisms on achieving the intended qualities of modifiability and scalability.

Any aircraft subsystem, independent of its detailed functionality, can be realized in simulation as a subsystem controller with one or more objects and an export area. From one subsystem to another, many things will vary. For example, the objects themselves and the exact manner in which they are coordinated will vary from subsystem to subsystem. Yet this basic structure—a subsystem controller with one or more objects and an export area—will remain invariant. Furthermore, this same basic structure also allows for the specification and implementation of subsystems that are unique to the simulation rather than being derived from the aircraft. In addition to data gatherers and distributors, this structure can be used to specify subsystems like control loading, aural cue, equations of motion, and surrogates that allow for partial device training.

The coordination mechanisms are sufficient to handle all modes of simulator operation and all events from external sources.

- The mode sequencer and its companion set of mode-specific ordered lists allow subsystems to be updated at different rates. By calling the subsystem's import procedure while the simulation is quiescent, the mode sequencer can frame data for subsystems that require a time-consistent view of inputs.

- The event handler is a sufficiently robust mechanism for mapping events to subsystems. Such events include malfunction setting and clearing, parameter sets and holds, and simulator mode changes. Subsystem controllers then map the events to the appropriate subsystem objects.
- The subsystems and objects themselves are able to implement all simulator modes and respond to all external events.
- Export areas are a sufficiently robust communication mechanism for communication between subsystems on one processor, subsystems on separate processors, and other external simulator systems such as the IOS, visual, and motion systems.

Modifiability and scalability are both positively affected by the partitioning strategy and the coordination and communication mechanisms. Characteristics of the structural model that contribute to modifiability and scalability include:

- Traceability between aircraft subsystems and simulation subsystems.
- Separation of computation from coordination.
- Localization of knowledge.

There is a well-defined locus of change for subsystems and objects that simulate aircraft subsystems and components. When an aircraft component is changed, the change is traced to the subsystem in which the component is simulated and its corresponding object is changed. As long as the interface to the subsystem (inputs from other subsystems and outputs to other subsystems) stays the same, the change is localized within the subsystem (subsystem controller and/or one or more objects). Even if the interface does change, the effects of the change to the aircraft subsystem can be traced by tracing the effects of the change to the interface. Thus, the partitioning strategy leads to the same kind of cohesiveness between software components as that shared by the physical aircraft components.

The separation of computation from coordination, a consequence of encapsulating models in objects, limits the number of assumptions that modelers can easily make, thereby reducing unnecessary dependencies between subsystems. Where temporal dependencies do exist, they can be resolved by exercising the mode sequencer's ability to frame data for a subsystem. The net effect is a reduction in temporal dependencies between subsystems that leads to simpler integration and greater scalability.

Another key contributor to modifiability and scalability is the localization of knowledge that comes from the partitioning strategy. Functional requirements for capabilities like reconfiguration, initialization, etc., are allocated to each subsystem rather than to individual entities. Thus, every subsystem localizes knowledge about how to initialize itself, how to update itself, what it means to reconfigure, how to stabilize computations following a reconfiguration, what malfunctions are simulated and how they are simulated, etc. This consistent localization provides mechanisms for coordinating the behavior of subsystems across simulator modes (ordered lists, event mappings) and simplifies the addition of new subsystems. Furthermore, localiza-

tion leads to well-bounded subsystems that provide an effective unit of distribution when balancing loads across processors. All of this contributes to scalability and modifiability.

3.4 Application of the Air Vehicle Structural Model

Again, the goal in applying a structural model on a project is to see that the many decisions reflected in the structural model are adhered to consistently. There are two aspects to this consistency:

1. Consistent transformations between the specification of subsystem controllers, objects, export areas, etc., and the implementation of the specified elements.
2. Consistent use of the structural elements to fashion simulation models.

As described in Section 2.4, consistent transformations can be accomplished by using specification forms and code templates. Specification forms allow simulation analysts to specify an arrangement of structural elements (subsystem controller, objects, and export area) for simulating particular subsystems, as well as details about what each of the elements contributes, what malfunctions are simulated, how the subsystems behave in different simulator modes, etc. Code templates allow the consistent realization of the structural elements specified on specification forms. Thus, there are code templates for subsystem controllers, objects, and export areas. Every instance of an object on a specification form, for example, eventually leads to instantiation of the code template for objects.

The consistent use of the structural elements to fashion simulation models will be more difficult, at least until simulation analysts acquire experience with the structural model. The structural model prescribes that an aircraft subsystem be simulated by simulating and coordinating the behavior of its component parts. In many cases, this is very different from what simulation analysts are accustomed to. Through years of conditioning and experience with the traditional software design for simulation, simulation analysts have become accustomed to viewing simulation problems functionally. Faithful application of a new structural model involves more than taking the old functions, placing them in object packages, and mechanically calling the "objectified" functions in order. It involves a thorough understanding of the role of the structural elements and the quality goals they are intended to support.

Specification and implementation of exemplary subsystems is one way of helping to reestablish the experience base. Exemplary subsystems (or **exemplars**, for short) are an effective means of demonstrating the intent and benefits of a structural model to simulation analysts. To the extent that exemplary subsystems are representative (functionally) of the subsystems that analysts must simulate, exemplars provide effective guidance by way of analogy.

Lack of experience with the structural model also erodes the basis for predicting both human and computer resources. **Incremental development** is one method of restoring the predictive basis. In incremental development, the engineer builds a subset of the subsystems to be simulated, then repeats the process for remaining increments. The consistent and repeated ap-

plication of a small number of structural elements for specifying and implementing subsystem simulations allows the experience gained from the first increment to be applied to subsequent increments. The results of each increment are run on the target computer, allowing early and ongoing insight into the use of computer resources. This process is largely independent of the particular subsystems built in each increment.

3.5 Summary

Modern-day intended qualities for flight simulators led to the development of an air vehicle structural model that emphasizes modifiability and scalability in addition to efficiency. This structural model is a product of an object-oriented partitioning strategy in which the software structure is based on aircraft subsystems and components rather than on functional requirements. Functional requirements for all simulator modes (initialization, run, freeze, etc.) are partitioned to each subsystem. Partitioning to subsystems localizes the effects of changes to software in response to changes in the aircraft and establishes the subsystem as an effective unit of distribution for balancing loads across processors. These characteristics contribute to satisfying the goals of modifiability and scalability.

The structural elements provided by the structural model (subsystem controller, objects, export areas, etc.) are sufficient for implementing all functionality within its scope. The coordination mechanisms are sufficient for implementing all required operational modes of the simulator, for coordinating computations distributed over several processors, and for coordinating interactions with other simulator systems outside of the air vehicle simulation (IOS, visual system, motion system, etc.).

The air vehicle structural model implies more than a new way of implementing old functionality. The structural elements impose a view of simulation that is different from the functional view imposed by the traditional software design for flight simulators. Thus, the air vehicle structural model implies not only a new way of packaging simulation models; it also implies a new way of thinking about simulation models that is driven by the goals of modifiability and scalability.

Initially, lack of experience with a structural model will inhibit its consistent, efficient, and predictive application. As the structural model is applied from project to project, an experience base will gradually be rebuilt. First-time users of the structural model can manage the change by developing (or borrowing) exemplary subsystems to serve as examples for simulation analysts and by developing a large simulator in small, well-planned increments. It is important for first-time users to realize that development (and acquisition) practices predicated on traditional designs are probably inappropriate in this context.

4 Structural Models in Proposals

Developing a structural model is an integral part of system engineering Once developed, a structural model allows system design decisions to be applied consistently, efficiently, and predictively throughout the development process. Applying a structural model on a flight simulator project involves repeated use of a set of structural elements to specify and implement simulations of individual aircraft subsystems. Thus, a structural model can be evaluated early in the development process, independent of the particular subsystems it will be used to implement. The evaluation includes an assessment of the proposed system's ability to provide required functionality and to deliver the qualities that it is intended to deliver.

A proposal to use structural models should communicate all that the bidder knows about the structural model(s) the bidder intends to use. ASC/YT assumes that, as the community becomes more familiar with structural models, the community will be able to provide more information in proposals. In the meantime, it is sufficient to establish certain system engineering goals and to identify when in the development process those goals will be satisfied.

Program specific details of what ASC/YT expects to see in a proposal will be addressed in the Instructions to Offerer (ITO). In general,a complete proposal should:

1. Describe the structural model(s) to be used by the project.
2. Demonstrate that the proposed structural model is complete with respect to required functionality and will deliver the intended qualities.
3. Describe how the structural model(s) will be applied consistently on the project.

In the remaining sections, we elaborate on ASC/YT expectations in each of these areas. For each, we first describe what a bidder who has extensive experience with structural models is likely to provide in a proposal. We then describe the minimum amount of information that could be provided by new users of structural models.

4.1 Description of a Particular Structural Model

A description of a structural model for flight simulators should begin with a statement of the scope of the structural model and a description of the intended qualities that drove the tradeoff decisions that were made when the structural model was formulated. For example, the scope of the structural model discussed in Chapter 3 is the air vehicle system, and the intended qualities are modifiability of the air vehicle and scalability. Other examples of the scope of a structural model include the IOS, the threat environment, and mission planning. Each example might have its own set of qualities that led to the particular model. The relative importance of the qualities should also be indicated.

The description should next identify the partitioning strategy. The partitioning strategy described in Chapter 3 is object-oriented decomposition. The intended results of the partitioning

strategy should be established by example; the examples should demonstrate what is meant by the terminology used to describe the partitioning strategy. For the structural model discussed in Chapter 3, one might list typical subsystems and indicate what objects will be simulated. Without such examples, the intent of the partitioning strategy is difficult to determine.

A description of a structural model should include both a static and a dynamic description of the elements of the structural model. The static description should explain both the role and the form of each element. For example, the role of the object element discussed in Chapter 3 is to encapsulate computations that simulate the behavior of an aircraft component. The form of the element might be an Ada package with private internal state data and procedures that operate on the object. The dynamic description should show how the elements are coordinated to provide system behavior and how the elements communicate with one another. This can be done with a set of thread diagrams showing how the elements interact in important system modes (run, freeze, etc.). Another set of thread diagrams might show how events are triggered and reacted to.

Hardware is an integral concern of the system engineering team. The structural model is predicated on assumptions about the computing platform. This is particularly true for coordination among elements on different processors or clusters of processors. These assumptions should be made explicit in the description of the structural model.

At a minimum, a proposal should identify all structural models by indicating their scope. Any proposal that doesn't provide full descriptions should establish the provision and review of full descriptions as an early milestone.

4.2 Evaluation of a Structural Model

A structural model can be evaluated in terms of its **completeness** and the qualities it is intended to deliver. A structural model is complete if it can be used to partition all required functionality within its scope, and if it can provide the mechanisms of coordination and communication required in all modes of system operation. One way to demonstrate that the structural model can be used to partition and coordinate all required functionality is by example. For an object-oriented structural model, subsystems such as fuel, hydraulics, electric, and pneumatic present similar partitioning problems: all involve objects that distribute loads through networks. Another group of subsystems—the radios, radars, and other transmitters and receivers—is also similar with respect to partitioning and coordination problems. In these cases, partitioning and coordination solutions for one subsystem in the group indicate the ability to solve these problems for all subsystems in the group. Thus, this form of completeness can be demonstrated by an adequate set of partitioning and coordination examples.

Completeness with respect to coordination can be demonstrated by describing dynamic threads. For example, in run mode, it might be necessary to coordinate communications between subsystems running at different rates. After a reposition, it might be necessary for subsystems to stabilize their computations. Thread diagrams also should be provided for all

significant events, such as malfunction setting and clearing, parameter sets and holds, and simulator mode changes.

A structural model is designed to satisfy certain quality goals implicit or explicit in the request for proposals (RFP). For example, if the RFP emphasizes modifiability, the bidder should propose indicators of modifiability (for example, the ability to respond to a component upgrade or the addition of a new radar set). The bidder then should describe the required modifications to the software in terms of the structural elements involved.

At a minimum, bidders should list both the completeness criteria and the quality criteria supported by each structural model. If the bidder has not proposed a complete description of a structural model, it will not be possible to demonstrate that the structural model meets the criteria or satisfies the goals. However, the proposal can commit the bidder to demonstrate that the criteria have been satisfied when the promised structural model is reviewed.

4.3 Application of a Structural Model

The most critical aspect of the application of structural models is its relationship to system engineering. Creation and development of the structural model is the responsibility of the system engineering team. The proposal should establish that a system engineering team will assume responsibility for the structural model at the start of the program and maintain responsibility for the structural model throughout the project.

Consistent, efficient, and predictable use of a structural model requires a development approach that proceeds in concert with the development of the structural model. A proposal to use structural models should address the following aspects of its application:

- Staffing plan and work breakdown structure.
- Resource estimation and traceability.
- Prototyping, incremental development, and integration.
- Tools and notations
- Documentation

Both the proposed staffing plan and work breakdown structure should reflect the evolution and application of the proposed structural models. For example, one cannot begin analysis of detailed functionality until the partitioning strategy has been well established and realized in a structural model and until exemplary subsystems are available for training analysts. Since work breakdown structures reflect the partitioning of the personnel and the allocation of functionality to work teams. they must be consistent with the partitioning and coordination strategy of the structural model. This means that work breakdown structures based on experience with traditional designs are not necessarily appropriate for structural models derived from an object-oriented partitioning strategy.

A proposal should justify software and hardware costs in terms of the structural model. All lines of code are not equal. Coordination runtime costs vary from structural model to structural model. Development costs vary according to familiarity with the partitioning strategy and experience applying the structural model.

A proposal should indicate how structural models will be used to take advantage of the benefits of prototyping and how they will be used to facilitate migration from prototype to integration in well-planned increments of development. The commitment to specify and implement all partitions using a single set of structural elements facilitates early generation of a prototype complete with realistic synthetic loads. The prototype can be used first to resolve basic issues of hardware/software integration and to refine estimates of runtime resources. As development proceeds incrementally, synthetic loads can be replaced by actual code, code which can be integrated with code already in place. Increments can be planned with risk management in mind. Early increments might be made up of exemplary subsystems. Subsequent increments might include subsystems with widespread impact on other subsystems or subsystems for which estimates are most uncertain.

Any proposed use of tools and notations should be consistent with structural models in general. Some structural models allow a straightforward mapping between design and implementation. As described in Chapter 3.4, specification forms and code templates are ways of establishing this mapping. Proposals should focus on tools that facilitate form entry and mapping from specification to templates (or their equivalent). Tools that require multiple representations of the same information should be avoided.

A proposal should indicate how the structural model will be efficiently documented. For example, a simulator built using the structural model discussed in Chapter 3 might have 300 subsystems, each with an export area and a controller. Since each subsystem coordinates its objects and communicates results of computations in the same general manner, such design information need not be repeated 300 times. Instead, the documentation of each controller and of each export area can refer to a reference description of the structural model for the common attributes of each subsystem.

At a minimum, bidders should propose the central role of a system engineering team in the development and application of structural models. They should delineate the aspects of the development process that will reflect consistent, efficient, and predictable use of structural models, and commit to resolution of details early in the development process.

4.4 Summary

Structural models allow early analysis of the viability of a proposed software design. Structural models can be used to help ensure that system requirements, both functional requirements and quality goals, are met consistently, efficiently, and predictively. Thus, information on structural models is vital in assessing the technical quality of a proposed solution.

ASC/YT understands that during the maturation process, it will be difficult for bidders to provide complete information about structural models and their use in proposals. However, ASC/YT does expect that bidders will treat structural models as more than another software engineering buzzword to be mentioned in passing. Even bidders who have had no experience with structural models can demonstrate their understanding by including the minimal information identified above.

Appendix A Glossary

Activation - The mechanisms by which the structural elements are made to cooperate to realize overall system functionality.

Aperiodic - Not occurring according to a regular schedule; opposite of **periodic**.

Code template - A template used to ensure the use of consistent coordination mechanisms. Code templates map directly to specification forms.

Communication - The paths and mechanisms by which structural elements share the results of their computations.

Completeness - The ability of a structural model to partition all required functionality within its scope and provide the mechanisms of coordination required in all modes of system operation.

Computation - Using available computing resources to achieve some purpose; in the case of flight simulators, the purpose of computation is to simulate the behavior of a real-world air vehicle's components.

Coordination - Activation and communication.

Efficiency - The extent to which software performs its intended functions with a minimum consumption of computing resources.

Event handler - Structural element that detects and responds to aperiodic events that originate outside the air vehicle, such as from the instructor/operator station (IOS). Typical events include simulator mode changes such as freeze, run, re-initialize, etc.; malfunction setting and clearing; and parameter setting or holding.

Exemplars - Model solutions for particular structural elements that demonstrate to simulation analysts the intent and benefits of a structural model. To the extent that they are representative (functionally) of the subsystems that analysts must simulate, exemplars provide effective guidance by way of analogy.

Export area - Structural element whose purpose is to provide a place for a subsystem controller to keep its outputs, so that these outputs are accessible by other subsystems and elements. There is exactly one export area associated with each subsystem controller.

Fidelity - The faithfulness of a simulation to the real-world situation being simulated.

Flight simulator - A device that trains pilots and crew members to control the behavior of an aircraft under a variety of circumstances.

Incremental development - A technique that emphasizes the overlapping, iterative development of design and implementation.

Intended qualities - Qualities, such as modifiability, that are not part of a system's functionality but that are important to the system. A few dominant intended qualities tend to guide decision making in the design of large software systems.

Mode sequencer - Structural element that effects periodic coordination. The mode sequencer coordinates subsystem controllers to yield overall simulator behavior in all required modes.

Modifiability - The ability to change software components in a simulation easily.

Object - An encapsulation of data and the services that manipulate that data.

Object-oriented - Partitioning strategy in which objects are based on real-world components and provide operations, or methods, for providing services.

Partitioning - The decomposition of large software problems into smaller subproblems and the allocation of these subproblems to software components that will solve them.

Periodic - Occurring according to a regular schedule.

Scalability - The ability of a software system to accommodate changes in scale.

Simulator mode - The task with which the simulator is currently concerned. Example modes include run mode (performing normal operation), malfunction mode (performing in response to a malfunction), or replay mode (replaying the actions of a simulation).

Specification form - A form used by simulation analysts to specify how individual subsystems will be simulated. A specification form constrains the simulation analyst to express solutions to simulation problems in terms of the structural elements and coordination mechanisms contained in the structural model.

Structural element - Software component that embodies decisions about coordination.

Structural model - A pattern for specifying and implementing software system functionality. It reflects system engineering decisions about partitioning and coordination. A structural model allows critical decisions about system design to be evaluated early in development and allows the decisions to be applied consistently across the entire software system.

Subsystem controller - Structural element whose job is to coordinate the computations performed by individual objects to yield meaningful subsystem behavior.

System - A collection of subsystems organized to accomplish a specific function or set of functions.